Implementation of a Probabilistic Neural Network for Multi-spectral Image Classification on an FPGA Based Custom Computing Machine

Marco A. Figueiredo
marco@fpga.gsfc.nasa.gov
SGT, Inc. / NASA Goddard Space Flight Center
Greenbelt, Maryland , USA

Clay Gloster
gloster@eos.ncsu.edu
Department of Electrical & Computer Engineering
North Carolina State University, Raleigh, NC, USA

Abstract

As the demand for higher performance computers for the processing of remote sensing science algorithms increases, the need to investigate new computing paradigms is justified. Field Programmable Gate Arrays enable the implementation of algorithms at the hardware gate level, leading to orders of magnitude performance increase over microprocessor based systems. The automatic classification of space borne multispectral images is an example of a computation intensive application that only tends to increase as instruments start to explore hyperspectral capabilities. A probabilistic neural network is used here to classify pixels of a multi-spectral LANDSAT-2 image. The implementation described utilizes a commercial-off-the-shelf FPGA based custom computing machine.

1. Introdution

A new generation of satellites is under implementation by the National Aeronautics and Space Administration (NASA) to compose the Earth Observing System (EOS). The instruments aboard the EOS satellites not only extend the observation life of the current system, but they also expand the capabilities of remote sensing scientists to better understand the Earth's environment. Along with the scientific advancements of the new missions, it is also necessary to explore new technologies that facilitate and reduce the cost of the data analysis process. In order to process the high volume of data generated by the new EOS satellites, NASA is constructing the Distributed Active Archive Centers (DAACs), an expensive and powerful parallel computing environment. Scientists will be able to request certain data products from these centers for further analysis in their own computing systems. A new technology that could bring the processing power to the scientist's desk is highly desirable. The ultimate scenario would be for the scientist to request the data directly from the satellite along with historic data from an archive center.

The advent of Field Programmable Gate Array (FPGA) enables a new computing paradigm that may represent the future for remote sensing scientific data processing. Several applications utilizing FPGA based computers have been developed showing orders of magnitude acceleration over microprocessor based systems [3],[4],[5]. Moreover, microprocessors and FPGAs share the same underlying technology – the silicon fabrication process. Therefore, it is reasonable to conclude that FPGA based machines will always outperform microprocessor based systems by orders of magnitude [6],[7].

The Adaptive Scientific Data Processing (ASDP) group at NASA's Goddard Space Flight Center (GSFC) has been investigating the utilization of FPGA based computing, also kown as adaptive or reconfigurable computing, in the processing of remote sensing scientific algorithms. The first prototype developed by the group utilized a comercial-off-the-shelf (COTS) reconfigurable accelerator in the implementation of an automatic classifier for the LANDSAT-2 multispectral images. The classifier algorithm utilizes a probabilistic neural network (PNN). The results show an order of magnitude performance increase over a high-end workstation. This paper presents details of the FPGA design and is organized as follows. Section 2 describes the PNN algorithm. The implementation of the FPGA custom computing machine is then presented. Finally, a performance analysis is given.

2 The PNN multispectral image classifier

Remote sensing satellites utilize multispectral scanners to collect information about the Earth's environment [8]. The images formed by such instruments may be seen as a set of images each corresponding to one spectral band. A multispectral image's pixel is represented by a vector of size equal to the number of bands. The combination of the multiple spectrum measurements represented by each element of the pixel vector determine a signature that correspond to a physical element. Through the observation of an image and the comparison of certain pixels to elements from known locations (in-situ measurements), a scientist is able to identify signatures and compose classes. These classes contain representations of multispectral pixels that are closely related. Several neural network schemes have been devised for the automatic classification of multispectral images [1]. One in particular, the Probabilistic Neural Network (PNN) classifier presented good accuracy, very small training time, robustness to weight changes, and negligible retraining time. A description of the derivation of the probabilistic neural network (PNN) classifier is given in Chettri et al., 1993 [2].

The same algorithm and data set used in [2] were used in this work to demonstrate the effectiveness of FPGA based computing. The Blackhills (South Dakota, USA) data set was generated by the Landsat 2 multispectral scanner (MSS). The image's 4 spectral bands (0.5-0.6 um, 0.6-0.7 um, 0.7-0.8 um, and 0.8-1.1 um) correspond to channels 4 through 7 of the Landsat MSS sensor. There are 262,144 pixels corresponding to a 512x512 image size, and each pixel represents 76m x 76m on the ground; the images were obtained in 1973. The ground truth was provided by the United States Geological Survey. Table 1, extracted from [2], shows the distribution of the image data.

	Training Number of Pixels	Entire Image Number of Pixels	Class name USGS – Level 1
0	453	6676	Urban
1	478	42432	Agricultural
2	464	16727	Rangeland
3	482	198868	Forested Land
6	368	1441	Barren

Table 1 - Distribution of data, Blackhills data set

Figure 1 illustrates the PNN classifier procedure. Each multispectral pixel, represented by a vector, is compared to a set of pixels belonging to a class. Equation 1 is used to derive a value that indicates the probability that the pixel fits in that class. A probability value is calculated for each class. The highest value indicates the class in which the pixel fits in.

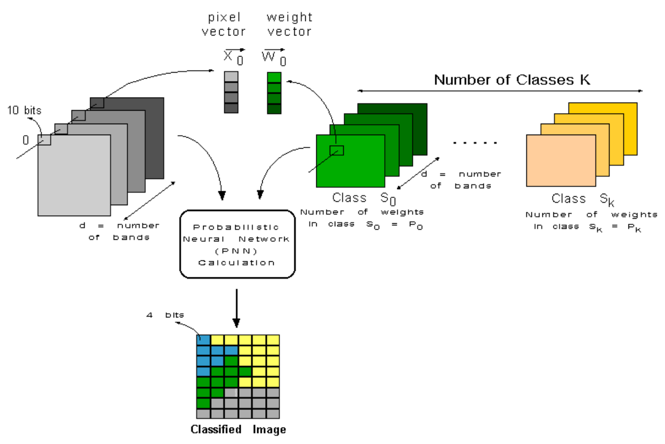

Figure 1 - PNN Image Classifier

$$f(\vec{X}|S_k) = \underbrace{\frac{1}{(2\pi)^{d/2}\,\sigma_k^d\,P_k}}_{K1[k]} \sum_{i=1}^{P_k} \exp\left[-\underbrace{\frac{1}{2\sigma_k^2}}_{K2[k]} \cdot (\vec{X}-\vec{W}_{ki})^T \,(\vec{X}-\vec{W}_{ki})\right]$$

$\vec{X}$ - Pixel Vector

$\vec{W}_{ki}$ - Weight i of class k

d - number of bands

k - number of classes

P_k - number of weights per class

Equation 1- PNN classifier

3 The FPGA implementation

Field Programmable Gate Arrays (FPGA) are logic devices that offer in-circuit re-programmability. Adaptive, or reconfigurable, computing is an emerging technology that utilizes FPGAs to implement computation intensive algorithms at the hardware gate level. As a result, acceleration rates of several orders of magnitude faster than current computers are attainable.

A set of FPGA devices arranged in some kind of programmable interconnection network is called a reconfigurable or adaptive computer. Current reconfigurable computers function like coprocessor cards which are plugged into desktop or large computer systems, called the host. By attaching a reconfigurable coprocessor to a host computer, the computation intensive tasks can be migrated to the coprocessor forming a more powerful system.

The first task in the implementation of an application is to select the adaptive coprocessor that best matches the algorithm in question. At the current state of the technology, there is no single COTS coprocessor card that will give best performance in most applications. A preliminary analysis of the PNN classifier indicated that the Giga Operations Spectrum System [9] presented the best architecture for its modularity and expandability.

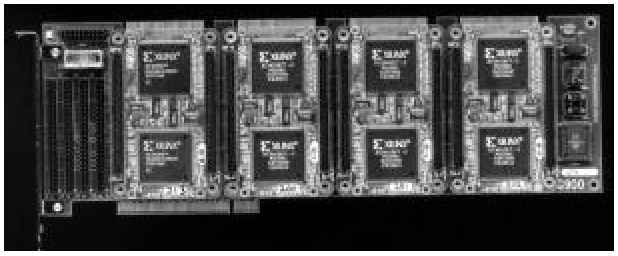

Figure 2 - G900 Spectrum System

The Giga Operations Spectrum System, shown in figure 2, is composed of a PCI bus based motherboard and 16 plug-in modules, 4 stacks of 4 modules each. These plug-in modules contain two Xilinx FPGA devices and provide the gate capacity based on the type of FPGA device being used. Our design was developed based on Xilinx XC4013E FPGA devices with an equivalent 13,000 gates per each device, or 26,000 gates per X-213 module.

3.1 Algorithm partitioning

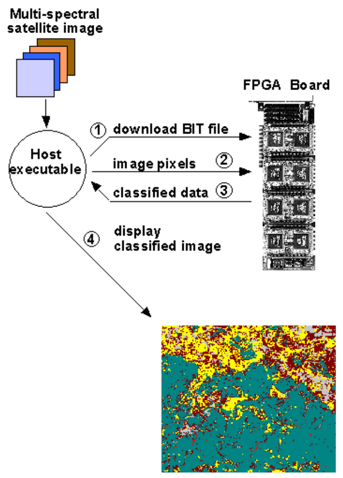

Figure 3 - Algorithm Partitioning

The computation intensive portion of the multispectral image classification algorithm resides on the calculations within the PNN classifier. The user interface, data storage and IO, and adaptive coprocessor initialization and operation is performed on the host computer. The PNN classifier was mapped to a single X213 module. Figure 3 illustrates the algorithm partitioning.

3.2 FPGA application design

Figure 4 shows the architecture of the X213 module. The YFPGA has a direct connection to the host through the HBUS. The memories between the two FPGAs were not used. An interconnection bus was used to transfer data from the X to the YFPGA instead. The HBUS was also extended from the Y to the XFPGA to allow the host to read back the results of the YFPGA. Only the SRAM banks on the XMEMBUS and HMEMBUS were utilized.

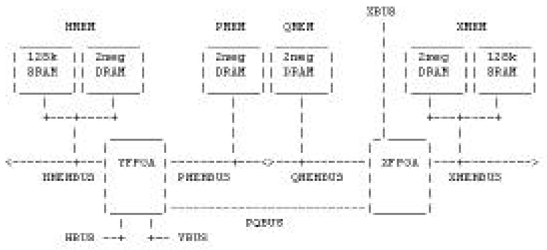

Figure 4 - X213 Block Diagram

Due to limited number of gates available on a single FPGA, it was not feasible to use floating point arithmetic in our implementation of the PNN algorithm. Hence we transformed the algorithm into fixed point prior to hardware implementation. The width of the fixed point data path was determined by simulating variable bit operations in C and comparing the results obtained from the original algorithm in floating point. Once the fixed point classification of the Blackhills data set yielded the same results as the floating point version, data path width for the FPGA implementation was no longer varied.

Figure 5 shows the data flow diagram for the hardware implementation of the PNN classifier. The number of bands (d) was fixed to 4, the maximum value of the number of weights per class (P_k) was fixed to 512, and the maximum number of classes (k) was set to 16. As shown in equation 1, there are two constants, K1 and K2, that are class dependent. These constants are pre-calculated on the host and downloaded to memory banks residing on the FPGAs.

The weights memory was mapped to the HMEM SRAM. Due to the lack of space on the XFPGA, the K1 multiplier and the class comparison blocks were moved to the host. These calculations amount to k (number of classes) multiplications and comparisons per pixel classification. Overall, they do not account for a significant amount of the computation, leading to a small performance penalty. A 4-bit register holds the number of classes. This register is initialized by the host before loading the FPGAs. The weight memory can be as large as 16*512*4*2bytes = 32768 16-bit words. The weight

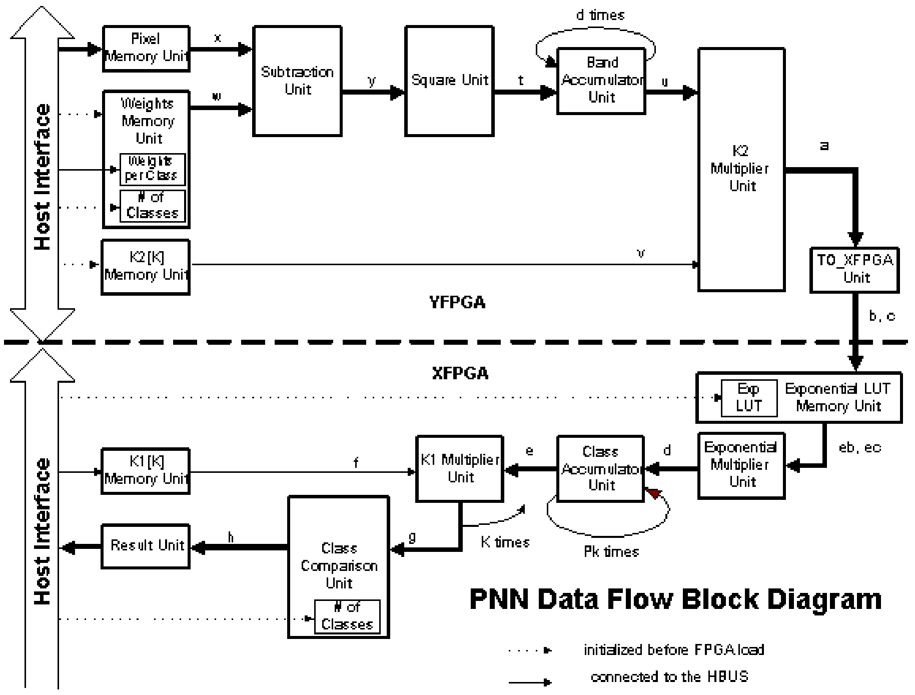

Figure 5 - PNN Data Flow Diagram

values are 10-bits wide. Since each class can have up to 512 weights, an array that holds the number of weights for each class is used. The arrays' data inputs are connected to the HBUS allowing visibility from the host application.

The Subtraction Unit subtracts W, a 4 x 10-bit element vector for W (w0, w1, w2, w3) and X (x0, x1, x2, x3). The result of the subtraction ranges from -1023 to 1023, requiring 11 bits to be represented in two's complement format. The Square Unit multiplies the 11-bit elements of the Y vector by themselves (t0 = y0 * y0). The values of the elements of vector T range from 0 to 1,046,529, asking for 20 bits to be represented in two's complement format.

The Band Accumulator Unit adds the 4 elements of the T vector delivering u, which value ranges from 0 to 4,186,116, requiring 22 bits. The K2[K] Memory holds the K2 values for each class. K2 = ½ σ_K^{-2} , where σ_K varies between 2 and 12, with increments of 1. As a result, K2 varies between 0.125 ($\sigma = 2$), and 0.003472 ($\sigma = 12$). The largest value of K2 = 0.125 is represented in binary by 0.001. In order to increase the precision of the multiplication, the values of K2 are stored with the decimal point shifted to the right by 2 (a multiply by four effect). After K2 is multiplied by u in the Multiplier 1 Unit, the decimal point of the result of the multiplication is shifted to the left by 2 (divide by 4 effect). Since this is a representation issue, no hardware is necessary to perform the shifts in the YFPGA, only the host needs to stores the values in the K2[K] memory in the above mentioned format. The K2 Multiplier Unit multiplies the K2 values for each class by the accumulated values of the difference between a pixel and a weight vector. It delivers a 44-bit result to the TO_XFPGA unit. Bits 0 to 23 represent the fraction portion (remember that the decimal point is shifted to the left by 2), and bits 24 to 43 represent the integer part of the result. However, the next operation is to extract the exponential of the negative of this number. Given the precision of the follow-on operations, any number above 24 will yield 0(zero) as a result. Thus, if any of bits 43 to 29 is set or both bits 28 and 27 are set, the result of e^{-x} should be zero. Only 28 bits are passed on to the Exponential LUT Unit, and they are bits 1 to 28. Bit 0 and bits 29 to 43 are discarded. It was also found that a considerable number of results of the multiplication are zero, which indicates that the result of the exponential should be one. In order to save processing steps in this case, the output of the multiplier is tested for zero, and a flag is passed to the Exponential LUT Unit, indicating that its result should be 1.

A look-up table is used to determine the value of e^{-a}. If we assume that a = b + c, then:

$$e^{-a} = e^{-(b+c)} = e^{-b} . e^{-c}$$

Since **a** is a 28-bit binary number, the value comprising bits 27 to 14 of **a** represent **b**, and the value comprising bits 13 to 0 of **a** represent **c**. The range of values of **b** and $\mathbf{e}^{-\mathbf{b}}$ are:

$$00000.000000000 <= \mathbf{b} <= 10111.111111111, \text{ or}$$
$$0 <= \mathbf{b} <= 23.9980469, \text{ which results in}$$
$$0.9980519 >= \mathbf{e}^{-\mathbf{b}} >= 3.78*10^{-11}$$

The range of values of **c** and $\mathbf{e}^{-\mathbf{c}}$ are:

$$00000.\ 000000000\ 00000000000001 <= \mathbf{c} <=$$
$$00000.000000000\ 11111111111111, \text{ or}$$
$$1.19*10^{-7} <= c <= 1.8919*10^{-3}, \text{ which results in}$$
$$0.999999881 >= \mathbf{e}^{-\mathbf{c}} >= 0.998109888$$

The values of $\mathbf{e}^{-\mathbf{b}}$ and $\mathbf{e}^{-\mathbf{c}}$ are previously calculated and organized into a look-up table. At run time, the values of **b** and **c** are used to address the look-up table stored in memory (XMEM SRAM). The values of $\mathbf{e}^{-\mathbf{b}}$ and $\mathbf{e}^{-\mathbf{c}}$ retrieved from the look-up table are then multiplied to give the value of $\mathbf{e}^{-\mathbf{a}}$. The values stored in the look-up table are 32-bit wide. The result of the multiplication is 64-bit, but only the most significant 32 bits are sent out. As a result,

$$3.77*10^{-11} <= \mathbf{e}^{-\mathbf{a}} <= 0.998051781.$$

The Class Accumulator Unit sums up all the comparisons between a given pixel and all weights of a given class, and outputs the result when it receives a flag indicating that the data to add to the accumulator refers to the last weight in a class. The output of the Exponential Multiplier Unit range is $3.77*10^{-11} <= d <= 0.998051781$. Thus, the largest accumulated value is $0.998051781 * 512$ (max. # of weights) = 511.002511872. In order to keep the precision of d, the accumulator is extended to 40 bits to accommodate the original 31 bits after the decimal point and 1 bit before the decimal point, and the new 8 bits before the decimal point. Each class has a K1 value associated with it. The value of K1 is determined by the following formula:

$$K1 = 1 / [(2\P)^{d/2}\ \sigma_{\kappa}^{d}\ P_k]$$

The result of the multiplication of K1 by the accumulated differences between a pixel and all weights in a given class is used to be compared with all other classes to determine the largest result, which indicates in which class a pixel most probably belongs to. In order to keep the values being multiplied in the same range allowing us to use fixed point arithmetic, the values of K1 are normalized. Given d, σ_{κ}, and Pk, the host program calculates all K1 values, and divide them by the largest one. The result is one value of K1 equals to 1 and all the others below 1. The K1 Multiplier Unit multiplies the 40-bit result of the Class Accumulator Unit by the 32-bit K1 value from the K1 Memory Unit, and outputs a 40-bit result to the g register in the Class Comparison Unit. The Class Comparison Unit receives a value that represents the comparison between a pixel and all weights in a class, and compares this value against the values generated for all other classes. At the end of the calculation of all classes, it outputs a code that represent the class which presented the largest value.

3.3 The host software

The software that was developed for the PNN algorithm that executes on the host processor was written in the Java programming language. We selected the Java programming language for several reasons. The language supports software reuse, native methods, remote method invocation, and it has a built-in security manager. Software reuse allows methods and other Java objects to be used repeatedly in different applications. Native methods allow legacy code (old software written in another language) to be called directly from Java methods. The security manager and remote method invocation allow methods to be executed on remote CPUs with the system taking care of network traffic errors, security, etc.

The Spectrum system, used for development of the hardware modules, contains drivers for interfacing to the FPGA devices that are only available in the C programming language. Java, was a very wise choice for a programming language since native methods, allow one to call C routines directly from Java. This is accomplished by building a dynamic link library that contains the C objects and calling these C functions directly from a Java method.

The application was implemented using a client/server methodology to facilitate future implementation of remote versions of the image classification algorithm. The server interfaces directly to the reconfigurable accelerator via the C drivers. It receives a block of pixels from the client, initiates the classification of each of the pixels on the fpga accelerator, gathers the results into a block of classified data, and sends the results back to the client. The client software controls the user interface, image data input/output and translation, in addition to communication with the server.

By selecting Java as a programming language and separating the program into client and server subsystems, the client software is completely independent of the operating system that will execute the client program.

Only the server contains code that is not only dependent on the operating system used, but also depends on the specific reconfigurable accelerator that has been selected. Hence, we are poised to implement a version of the PNN algorithm that can be executed on a remote machine found anywhere on the Internet.

4 Performance analysis

An initial version of the PNN Classifier algorithm was developed and executed on a 200MHz single processor Digital Equipment Corporation (DEC) Alpha workstation. This implementation, written entirely in C, required 12 minutes to classify the complete Blackhills data set. We also implemented the same algorithm on a Pentium PC running at 166 MHz. An optimized version of the algorithm implemented in Java, required 21 minutes 57 seconds, while a C implementation of the algorithm that was called from a Java method required 30 minutes 17 seconds. Please note that a Java implementation of the algorithm performs very well even though the language is interpreted.

By augmenting the PC with the Spectrum System G-900 with a single X213 module running the PNN classifier, the processsing time was reduced to 2 minutes 21 seconds (an acceleration of 9.34 times that of the Java software version.) By adding a second X213 module, 2 pixels are processed at the same time. The image classification time is then reduced to 77 seconds yielding a speedup of 17.10 over the Pentium PC alone, and 9.35 times faster than the DEC Alpha. This is approximately twice as fast as the single module version. Since the pixel classification does not depend on neighboring pixels, our implementation is easy to extend to several processing elements if hardware resources are available. We expect that with the addition of more X213 modules, the acceleration rate can be increased until the problem becomes IO bounded (CPU time required for input/output dominates total execution time.)

Also, the FPGA implementation is not fully optimized since the complete algorithm was not incorporated into the FPGA. Higher performance could be achieved if the K1 multiplier, the class comparator, and a FIFO memory were included on the FPGA. With the addition of these components, computation and data transfer could occur simultaneously.

5 Conclusions

It was shown that the implementation of a probabilistic neural network multispectral image classifier on an adaptive computer yields an order of magnitude performance increase over high end workstations. It was also shown that the combination of Java and an adaptive computer presents a potential solution for the remote processing of computation intensive scientific algorithms. The two technologies enable a new computing paradigm of distribuited, and reconfigurable data processing.

The disadvantages of FPGA based computing lie mostly on the maturity of the technology. The application development process requires new thinking. The FPGA devices themselves have not yet achieved the density – number of gates – and the flexibility to enable a larger set of applications. The development tools are either hardware or software oriented, and do not fully explore the capabilities of FPGA based computing. As a consequence, a wide set of skills are required to design applications that fully exploit the benefits of the new technology. As the technology develops, however, better tools and FPGA architectures will be available to enable a new class of application designers.

References

[1] S. R. Chettri, R. F. Cromp, M. Birmingham, “Design of neural networks for classification of remotely sensed imagery”, Telematics and Informatics, Vol. 9, No 3, pp. 145-156, 1992.
[2] S. R. Chettri and R. F. Cromp, “Probabilistic neural network architecture for high-speed classification of remotely sensed imagery”, Telematics and Informatics, Vol. 10, No 4, pp. 187-198, 1993.
[3] P. Athanas and L. Abbott, “Real-Time Image processing on a Custom Computing Platform,” IEEE Computer, Vol. 28, No 2, pp 16-24, February 1995.
[4] N. Shirazi, P. Athanas, and A. Abbott. “Implementation of a 2-D fast fourier transform on an FPGA-based custom computing machine”, Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications, pp. 282-292, FPL August/September 1995. Lecture Notes in Computer Science 975.
[5] M. Rencher, B. Hutchings, “Automated Target Recognition on Splash-II”, IEEE Symposium on Field Programmable Custom Computing Machines, pp 232-240, April 1997.
[6] N. Tredennick, “Get ready for reconfigurable computing”, Computer Design, pp 55-63, April 1998
[7] R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger; U. Kaiserslautern, “On Reconfigurable Co-Processing Units”, Proceedings of the 5th Reconfigurable Architectures Workshop (RAW'98) March 30, 1998
[8] D. L. Verbyla , “Satellite remote sensing of natural resources”, Lewis Publishers, 1995
[9] Giga Operations Corporation, “Spectrum Reconfigurable Computing Platform Documentation”